TRIAL SIZE TOOLBOX MANUAL

COPYRIGHT 1985
ROGER WAGNER PUBLISHING, INC.

DIVERSI-DOS (TM): COPYRIGHT 1982 DSR, INC.

INTRODUCTION

WELCOME TO THE TOOLBOX SERIES!

The Toolbox Series is the beginning programmer's dream, but also appeals to intermediate as well as advanced Applesoft programmers.

The Toolbox Series is a library of nifty commands which make writing ANY program in Applesoft quick and simple. It is viewed by many people as the "Applesoft Construction Set" because it allows you to build the programs you want out of an extensive set of "program building blocks" called Toolbox Files.

The Toolbox Series makes putting new commands and functions in your programs quick and easy. You won't need to find, research, write or debug these essential parts of your programs. For example, with the Toolbox, if you need to sort a list of names, all you do is let the Toolbox add a command to Applesoft to sort a list! The new sort command would look something like this:

```
100 & "SORT", A$(FE) TO A$(LE)
```

and in one statement the list would be sorted.

How does it work? That's the nice thing about the Toolbox Series; just decide what your program needs to know, use the Toolbox "Workbench" to teach Applesoft how to do it, and in the blink of an eye, Applesoft will have a new command that does what you need!

The Toolbox libraries that make up the series each contain from 25 to 40 different command subroutines, written in machine language, that can be added to any Applesoft program. The Chart 'n Graph Toolbox package has one large subroutine that adds 27 new graphics commands all at once to Applesoft and other individual commands that can be added one at a time as you need them. And, if you have an Apple //E with 128K or an Apple //C, the Chart 'n Graph Toolbox will give you all the new commands you need to use the DOUBLE HI-RES GRAPHICS capabilities of these machines for even more detailed graphics and charts!

The Toolbox Series has many other libraries in it as well. The Database Toolbox has over 40 array-related commands that are useful in any program that uses lists or Applesoft arrays; The Video Toolbox has over 30 text screen input and output commands; The Wizard's Toolbox has over 30 assorted commands including PRINT USING, SOUND EFFECTS and a HI-RES TEXT command. Try one, and you'll NEVER program without your Toolbox again!

This "Trial Size" version of The Toolbox will give you an idea of what The Toolbox Series can do for you, and let you use sample commands from the various library disks in your own programs right now.

Let's see how The Toolbox works.

A SHORT DEMONSTRATION

The primary purpose of the Toolbox Series is to add new commands to your own Applesoft programs, as easily and efficiently as possible.

The Toolbox uses a utility called the Workbench to actually add the new commands to your programs. The Workbench is located on side 1 of your Toolbox "Trial Size" diskette. All you have to do is:

1) Insert one special line in your program which tells Applesoft you have added some special new commands to your program.

2) Use the Workbench to add the commands of your choice.

3) Use these commands from within your Applesoft program by using the ampersand followed by the name of the new command.

To see how this works, let's consider an actual example. To start, first boot on side 1 of the Toolbox diskette, and press ESCAPE to exit the main menu to Applesoft BASIC.

Let's suppose for a moment that you would like to have some kind of musical sounds in a program you are writing. The Toolbox diskette has a file called TONE.TB which produces musical tones. Adding this command to your program is very simple, and the entire process can be done in less than a minute:

1. Type in FP to clear any program in memory.

2. With the Toolbox (side 1) in the drive, type in EXEC AMPERSAND SETUP. Then type in LIST and you should see:

   ```
   1 CALL PEEK(175) + 256 * PEEK(176) - 46
   ```

   When your program is run, this line will tell Applesoft that you have added various new commands.

3. Get the Workbench up and running by typing in:

   ```
   BRUN WORKBENCH
   ```

   After a few seconds, this menu will appear:

   ```
   *** TOOLBOX WORKBENCH ***

   SELECT AN OPTION:

   1. ADD A COMMAND
   2. REMOVE A COMMAND
   3. REMOVE ALL COMMANDS
   4. REPORT COMMANDS ADDED
   0. EXIT
                    [NO COMMANDS ADDED]
   ```

4. Since the first thing we want to do is to add a new command to our program, press '1'. The screen will then display:

INSERT DISK CONTAINING TOOLBOX FILE TO BE
ADDED, THEN ENTER NAME OF FILE

('CAT' FOR CATALOG, <RETURN> TO QUIT)

TOOLBOX FILENAME ->

If you knew that TONE.TB was the Toolbox File for the command you wanted to add at this point, you could just enter the name directly. Often you will not know the name exactly, in which case you can enter 'CAT' to produce a catalog of the diskette. Enter 'CAT' now. The disk will CATALOG and you should see TONE.TB. Now press RETURN to stop the catalog (assuming that you have booted with the Toolbox diskette).

5. Now type in TONE.TB as the Toolbox File to be added, then press RETURN. Note that all Toolbox module names have a '.TB' suffix.

   Next, the screen will display:

   YOUR COMMAND NAME ->

6. You must now enter the new name which you will use for this command within your program. Because the Toolbox Files themselves may have rather long names (so as to be most descriptive) you will often want to use a shorter name when using the command from within your program. In this case, 'TONE' will do nicely. Type in TONE as the name and press RETURN.

   The Workbench will now load the TONE.TB Toolbox File, and add the command 'TONE' to your program.

   After the command is added, the program will return to the request for a new FILENAME. This is because usually you will want to add several commands at one time. Press RETURN alone to return to the menu at this point.

7. Now enter '0' and press RETURN to exit the Workbench and return to your program.

8. When using these new commands in your Applesoft programs, a special 'syntax', or command structure, is required. Such a line has either two or three parts, as follows:

The first part required is the ampersand symbol (&). You can think of this as being similar to the Control-D character needed whenever a disk command is done. The difference here is that the ampersand will call a Toolbox command.

The second item required is the new command name (in quotes) for the command you wish to use. For our example, this would be "TONE". If the command which you have just added does not require any information to be passed to it, then nothing more is required in the command.

The third item is optional, and is referred to as a "variable list". Each Toolbox command may or may not have certain pieces of information which have to be passed to it. In the case of the TONE.TB command, these variables are pitch and duration. (In the case of other commands, the variable list following the name may have a different number of variables required and use a variety of variable types.)

To use your new command in a program, you would first determine the values for pitch (P) and duration (D), such as through an INPUT statement, and then use the new command (remember the ampersand!). To show how this might be done, enter the following program lines (these will now be in addition to the line #1 which was previously entered via the AMPERSAND SETUP procedure above):

```
10 INPUT "ENTER PITCH, DURATION: "; P, D
20 & "TONE", P, D
30 PRINT: GOTO 10
```

9. Now just RUN the program to try it out! Enter:

```
10,10
50,10
100,20
```

as some sample number pairs. As a point of interest, the duration is measured in 100ths of a second; thus a value of '100' gives a tone about 1 second long.

By the way, remember that, like normal Applesoft, you can use any variable names you want in using a command. As long as you include the TYPE of variables that the command expects, the name of the variable (or expression) is up to you.

YOU'RE DONE! You now have a complete program. The program can be LOADed or SAVEd just like any normal Applesoft program, and the new commands will automatically be carried along with it. They will remain a permanent part of your Applesoft program, unless you decide later to remove them using the Workbench "Remove a Command" option.

If you want to add more commands later, just BRUN WORKBENCH again, and add the commands you want. See the sections later in this manual to determine the exact syntax for using each command.

When developing a program using Toolbox commands, you can add new commands at any time while you are writing the program. Some people like to add most of the commands right at the start; others just add the commands as they need them while writing the program.

If It Doesn't Work...

In order for your program to use a new command it must meet two conditions: The ampersand link (line #1) must have been set up before any Toolbox command is used in your program, and there must actually be an added Toolbox File with the command name used in your program. If either of these conditions is not met, your program will crash unceremoniously.

THE AMPERSAND LINK. It is good practice, when developing a program which will use Toolbox commands, to BEGIN by EXECing the AMPERSAND SETUP file. As explained earlier, this inserts a line (line #1) in your program which tells Applesoft that you are using the Toolbox commands.

This line can occur anywhere in your program, provided that it is executed prior to any Toolbox command. If it is not present then you'll probably get a SYNTAX ERROR when you try to use one of your new commands. If this happens when you run your program, check to make sure the ampersand setup line is present AND being executed prior to the Toolbox command.

THE TOOLBOX FILE. Another way to bomb your program is to attempt to use a Toolbox command which has not been added. In this case (assuming that the proper setup line has been inserted) your program will try to find your Toolbox command name (following the ampersand - &) among the commands added. When it doesn't find it, you will get an UNDEFINED FUNCTION error message (with the line number). You can then determine which command is missing.

If you are sure that both the ampersand link and Toolbox File are present, then double-check the syntax for the routine (and perhaps the sample program listing given for each individual routine) as listed in the section on the Toolbox Files in this manual.

IMPORTANT NOTE: The Workbench may be used to add ANY ampersand-compatible machine language routine to an Applesoft program, and is not limited to the routines available on this or other Toolbox disks.

Adding More Routines/Resuming Normal Operation

If you want to add a new command right away, type in CALL 2051 to re-enter the Workbench. If you're done using the Workbench for a while and want to free up the memory normally occupied by the main Workbench utility, type in: BRUN REMOVE WORKBENCH. This will remove the Workbench utility from memory.

The Workbench utility is required in memory only when you are adding or removing commands (or doing any of the other things that it allows you to do). It is NOT required to be in memory when you run your program.

Because Hi-Res graphics cannot be used with the Workbench utility in memory, you should always remove the Workbench by typing in:

BRUN REMOVE WORKBENCH

before running any program that uses Hi-Res graphics.

Another way to put your program back where it belongs is first to SAVE it on disk, enter 'FP' and then LOAD your program.

You've now learned most of what it takes to use and enjoy the Toolbox system. The next sections describe a few other Workbench options, give a little detail about how the Workbench and Toolbox system use memory. The last part of this manual has descriptions of the sample Toolbox commands provided on this diskette.

OPTION #2: REMOVING ADDED COMMANDS

Suppose you have added 23 commands and then decide you really don't want command #5. The Workbench allows you to remove it easily. Select '2' at the menu, and the first page of a command report will appear. This report will show all of the added commands, giving both the command name, and the name of the Toolbox File from which the command was derived.

If you see the command you wish to remove, press the 'R' key to tell the program you wish to remove an item. Then enter the number of the command you wish to remove.

The Workbench will then remove the specified command and return you to the main menu. The other commands will not be affected and if you request a Command Report (see Option #4), you will find all the commands intact, except for the one just removed.

OPTION 3: REMOVE ALL COMMANDS

It is also possible to remove all added commands at one time. Select option #3 for this function. You will be asked to confirm your intentions for such a drastic action. You must respond with either 'Y' or 'N' to this question.

After removing all commands, if you ask for a Command Report (option #4), the program will beep to signify no commands present. Additionally, you may notice that next to the EXIT option NO COMMANDS ADDED is displayed in inverse.

OPTION #4: REPORT COMMANDS ADDED

This "Trial Size" version of the Toolbox Series may be used to add up to 3 commands to an Applesoft program. The standard version can add up to 255 commands to an Applesoft program. Although it is unlikely you will ever approach this number, you may from time to time wish to see a list of the commands which have been added to a particular Applesoft program.

Pressing '4' at the main menu will produce a Command Report (provided that there are any commands to report on). This is useful because it allows you to see what commands have already been added and to check on the names given to them.

When selecting the 'report' option, you are asked if output is to go to the printer. If you simply want screen display, press RETURN (or 'N').

If you want a print-out, press 'Y'. The Workbench will then ask:

PRINTER IN SLOT: 1

If this is the correct slot for your printer, press RETURN, otherwise enter the slot your printer is in (the Workbench will remember).

For each command added you are informed of the command name and the name of the Toolbox File from which the command was originally derived.

For example, using Option #4 of the Workbench might give the following information:

```
 ***  COMMAND REPORT  ***        [3]

COMMAND NAME:     FILE NAME:

#1  MENU          MENU GENERATOR.TB

#2  INPUT         STRING INPUT.TB

#3  GET           GET.TB

PRESS <RETURN> FOR MENU
```

With the Command Report, you can determine what Toolbox commands have been added to any Applesoft program. In fact, on the Trial Size Toolbox diskette, you'll find that a number of the demonstration programs also use Toolbox commands which are not specifically provided on this Trial Size diskette. A good example of this is the MENU command. This command creates the menu you see in the MENU program in just one Applesoft statement! MENU is supplied in the VIDEO TOOLBOX package, also available from Roger Wagner Publishing.

THE TOOLBOX CONCEPT - A LITTLE MORE DETAIL

Memory organization

Now that you have an idea of what the Toolbox does, here are some additional details of what actually goes on to establish the connection between the desired command and your Applesoft program. Although it is not necessary to understand all the information provided here to use the Toolbox, it may help you get a better feeling for what you are actually doing when you use the program.

When you first specify the name of a Toolbox File to be put in your Applesoft program, the Workbench does the following things:

1) It adds the Toolbox File to the end of your Applesoft program. Because of the nature of Applesoft, this code is not visible during a LIST, and is not affected by adding or deleted normal BASIC lines or statements.

2) It saves both the name you give to use the routine, and the name of the Toolbox File loaded. This is to make later identification of commands easier (see the section on the Command Report).

3) The first time a command is added, the Workbench adds its own 'Interface Routine' to the end of your program. The function of this is to locate a Toolbox File named in a command and to pass control to it.

The manner in which memory is normally allocated in the Apple is as follows:

```
-------------------------------------------------------
|     |                                       |   |    |
|FP   |<- - - - - Variables - - - - - - - - ->|DOS|ETC.|
|PROG.|                                       |   |    |
|     |                                       |   |    |
-------------------------------------------------------
^$800 ^LOMEM                                  ^HIMEM
```

Figure 1: "Normal" Applesoft program

Normally an Applesoft program resides at the "bottom" of memory, with all remaining space above it reserved by DOS and for variable storage.

The Workbench could be loaded right after your program, but as commands were added to the end of the Applesoft program, they would start to conflict with the Workbench itself. To avoid this, the Workbench moves your program UP in memory, and locates itself at memory location $800 (the dollar sign indicates a hexadecimal address).

```
-------------------------------------------------------
|       |     |       |      |                |   |    |
|WORK-  |FP   | T. B. |INTER-|                |DOS|ETC.|
|BENCH  |PROG.| FILES |FACE  |                |   |    |
|       |     |       |      |                |   |    |
-------------------------------------------------------
^$800   ^$2600                ^LOMEM          ^HIMEM
```

Figure 2: With the Workbench loaded

With your program safely relocated to $2600, Toolbox commands are automatically placed between the Interface Routine and the end of your program ('FP PROG' on the chart). The Interface Routine is a short routine that looks at the name following the ampersand each time you use an ampersand statement, and decides which of the added commands you actually want to use. When your program is run, the Line #1 that was put in by the AMPERSAND SETUP file points the ampersand vector to the Interface Routine at the end of your program. From there the name following the ampersand is read and the proper routine used.

CAUTION: Note that your program now STARTS at $2600. This is well into the Hi-Res page 1 display, which normally starts at $2000. Thus, any program using Hi-Res graphics will destroy itself if you attempt to run it with the Workbench itself in

memory. The Workbench should be removed before running any program using Hi-Res graphics and Toolbox commands

```
-------------------------------------------------------
|     |       |      |                       |   |    |
|FP   | T. B. |INTER-|                       |DOS|ETC.|
|PROG.| FILES |FACE  |                       |   |    |
|     |       |      |                       |   |    |
-------------------------------------------------------
^$800                ^LOMEM                  ^HIMEM
```

Figure 3: At 'Run Time'

If you use the REMOVE WORKBENCH utility, or type 'FP' and then do a fresh load of an Applesoft program containing added commands, then memory will be allocated as in Figure 3. This shows the Applesoft program back at its normal location, with the added routines and the Interface Routine still present.

Hi-Res graphics on page 1 will in most cases now be available. When in doubt, you can use the Memory Map option available in the standard version of the Workbench, which shows the addresses for your program at its normal location of $801.

When developing a program using Toolbox commands, remember that you can add new commands at any time while you are writing the program. Some people like to add most of the commands right at the start; others just add the commands as they need them when writing the program.

DEMONSTRATION PROGRAMS

Demonstration programs which use the Toolbox commands are provided on this "Trial Size" version of the Toolbox. These demos vary in their complexity, and may be listed and studied for more insight on how to use the Toolbox system in your own programs.

>> THE 30 SECOND TEST <<

If you are working on a program, and suddenly decide you would like to use a Toolbox command in it, you can add the command in less than 30 seconds. To prove this to yourself, try the following test (and have somebody time you!):

The setup:

You should have booted on the Trial Size diskette and exited to BASIC. Its disk operating system (DOS) is a special one (Diversi-DOS) that loads programs about 5 times faster than normal DOS 3.3, and can be used while you work on any Applesoft program. Next, type in NEW, and then a line to represent a program you might be working on. For the test, you can type in (after NEW):

```
0  REM MY TEST PROGRAM
```

The Test:

We'll pretend that you decide you want to add the TONE.TB routine used in the earlier examples. To add the command you can just type in the following (are you timing?):

1. BRUN W= (installs the Workbench)
2. Press '1' (selects 'Add a Command')
3. Type in 'TONE.TB' as the Toolbox file to add.
4. Type in 'TONE' as the new command name.
5. Press RETURN to go to main menu, and '0' to quit the Workbench.
6. Type in 'EXEC A=' (Installs line #1 in your program)
7. Type in 'BRUN R=' (Removes the Workbench)

THAT'S IT! You have just added the new command TONE to your program, and the command can now be used anywhere in the program with the simple statement &"TONE",P,D. How'd we do on the 30 second test?

SPECIAL TIP: The Fast Run Method

Although a good typist can meet the 30 second test even typing in the complete file names, anyone can use the Toolbox with the special commands just described because the special DOS that comes on the Trial Size Toolbox has been modified to recognize a "wild card" in the file name of a normal DOS command.

When typing the name of a file to be loaded or BRUN, etc., you can use just the first few letters of the name followed by an equal sign ("="). For example, if you wanted to BRUN the Workbench, you could just type in: BRUN WORK=

This tells DOS to BRUN the first file in the catalog starting with the letters 'WORK'. The '=' is called a wild card because it can represent any combination of letters at the end of a name. Because the Workbench, Ampersand Setup and Remove Workbench files are the first files on the diskette catalog, specifying just the first letter of each name followed by and equal sign is sufficient to run the appropriate file.

Other Special DOS Features

The Workbench DOS also has a few other extra features. These are a display of the number of free sectors remaining on a diskette when it is CATALOG'ed (an empty diskette has about 500), the option of pressing RETURN at any of the CATALOG pauses to stop the CATALOG listing, and the ability to use just the phrase 'CAT' to catalog a diskette.

>> THE TOOLBOX FILE LIBRARY <<

The following pages detail each of the commands provided on the "Trial Size" Toolbox diskette. These commands were selected from various Toolbox Series diskettes, each of which contain many new commands for your Applesoft programs. Ask your dealer, or write Roger Wagner Publishing for more information on these packages.

Each library entry describes both the function of the command and the syntax expected when using it. Syntax is shown using the conventions defined in the Applesoft Reference Manual, pages 144 to 149. Although these abbreviations may seem a bit stiff at first, the goal was for accuracy and clarity in presentation.

In addition to the formal syntax definition, several examples are usually given showing actual program lines using the given command.

The length of each command is also shown, to give you some idea of what memory is involved in implementing the command. For example, FIXED LENGTH INPUT.TB has a length of 490 bytes. This is considerably shorter than any alternative written in Applesoft, not to mention more efficient. The lengths should be checked by BLOADing the actual Toolbox file for any application where an exact length is critical, as the lengths may vary slightly from the documentation in the event of future revisions.

Note that the Pointer Read command introduces a new capability to Applesoft, namely, hexadecimal numbers. Routines that allow hex numbers expect the number to be entered directly (not in string or numeric variable form). Syntax expressions involving hex numbers will always show these numbers as 'hexnum'. A hex number is defined as a dollar sign ($) followed by 1 to 4 hex digits.

UNDOCUMENTED COMMANDS

We use the Toolbox ourselves in preparing virtually all of our commercial programs, including the menus and demos on this diskette. Be sure to use the Workbench to look at the Toolbox commands added to all the BASIC programs included on this diskette, including HELLO and MENU.

You'll find all sorts of interesting commands from the Toolbox Series that made life easier for us when this disk was created! Although these commands are not individually provided on the Trial Size Toolbox, they are available in other Toolbox packages.

>> ARRAY REDIM.TB <<

by Steve Cochard

FUNCTION: This command allows a previously dimensioned array to be redimensioned WITHOUT any loss of data. The command will work with integer, string and real arrays. It will redimension one and two-dimensional arrays only.

LENGTH: 978 ($3D2) bytes

SYNTAX: & "NAME", array name(aexp,aexp)

SAMPLE:
```
& "REDIM",A(10)
& "REDIM",A$(10,10)
& "REDIM",X%(A,10)
& "REDIM",A(5,C%(3))
```

HOW TO USE: This command will redimension any type of array without loss of data. It is used in the same manner as an Applesoft DIM statement. The subscripts specified in the command may be any arithmetic variable, constant or expression (real or integer).

This command should be used whenever it is determined, by a running Applesoft program, that a larger or smaller array is required. It could be used in an error handling routine that deals with BAD SUBSCRIPT errors. If such an error is encountered, the array in question is simply redimensioned to a larger size and control is returned to the point where the error occured.

LIMITATIONS: This command will not work with arrays that have more than two dimensions. This command will not convert one-dimensional arrays to two, nor two to one.

ERRORS: OUT OF DATA ERROR will occur if the desired array has not been previously dimensioned. A BAD SUBSCRIPT error will occur if a one-dimensional array is to be redimensioned to a two-dimensional array or vice versa. This error will also result if an array of more than two dimensions is used.

>> TONE.TB <<

by Craig Peterson

FUNCTION: Generates a pure tone of a given pitch and duration. Can also be used to pause.

LENGTH: 56 bytes ($38)

SYNTAX: &"NAME" [,pitch [, duration]]
&"NAME" [,aexpr [, aexpr]]

SAMPLE: &"TONE"
&"TONE",P
&"TONE",P,D
&"TONE",P+5,D/2

HOW TO USE IT: The TONE.TB routine can be used in a variety of ways. If no variables are included after the name, then a tone having about the same pitch and duration of the normal Apple beep will be sounded.

If one variable is included after the command name, it will change the pitch of the tone. This variable can be any numeric constant, variable or expression, but must have a value from 0 to 255. Lower values cause higher pitches. A value of '0' produces no sound. If the pitch value is doubled, then a tone about one octave lower will be produced.

If a second variable is given, the length of the tone can be changed. The length variable can also be any numeric constant, variable or expression having a value between 0 and 255. This length value is determined in approximately 1/100's of a second, so '100' would produce a tone of about one second in length. This allows tones with lengths ranging from 1/100 to over 2.5 seconds in length.

With pitch set to '0', the duration can be adjusted to produce silent waits of 1/100 to 2.5 seconds.

Be sure to run the TONE DEMO to see what the musical possibilities of this routine are.

LIMITATIONS: Both pitch and duration values must be numeric variables in the range of 0-255.

SAMPLE LISTING:

```
 1 CALL PEEK(175) + 256 * PEEK(176) - 46
10 INPUT "PITCH, DURATION?";P,D
20 &"TONE",P,D
30 GOTO 10
```

Here is a table of the note values produced with different values for pitch:

NOTE	PITCH NO.	PITCH NO.	PITCH NO.
F SHARP	135	67	33
F	143	71	35
E	151	75	37
E FLAT	160	80	40
D	170	85	42
C SHARP	180	90	45
C	191	95	47
B	202	101	50
B FLAT	214	107	53
A	227	113	56
A FLAT	241	120	60
G	255	127	63

AND.. 31 (G)

>> CHART TUTORIAL <<

This section provides a tutorial on how to use CHART.TB and what some of its more often used commands are. More information on each of the specific commands is found later in this manual.

GETTING STARTED

CHART.TB is a miniature version of a much larger graphics command set found in the Chart 'n Graph Toolbox package. The Chart 'n Graph Toolbox is an extraordinarily powerful graphics plotting and charting package designed to enhance the graphics capabilities of the Apple II family of computers. It is a collection of many graphics commands that will greatly simplify the task of using the Apple's Hi-Res graphics for the presentation of your data from within your Applesoft program. These commands are like a library of graphics utilities that act as an extension to the existing set of Applesoft BASIC commands and functions.

These commands will, for example, allow you to select which portion or "window" of the Apple screen you will be plotting your data. They will allow you to redefine the Apple coordinate system so that you are no longer required to adjust your data from 0-279 and 0-191. You could decide for example that the left side of your selected window is -1000, the right side is +10, the bottom is 0 and the top is 500. Then when you plot your data, if it's within your window's limits, you'll see it. If it's outside the window, it will be automatically "clipped", thus avoiding the usual Applesoft ILLEGAL QUANTITY ERROR.

Commands are available for the generation of axes with tic marks and grid lines, and of course you have the ability to produce alphanumeric labels with a variety of positioning options.

This tutorial is a collection of examples and techniques using the CHART.TB command set that should give you a feeling for the power that the Chart 'n Graph system affords the user in the presentation of graphical data in various formats. CHART.TB has a total of 12 graphics commands that you can start using in your programs right away.

See the command reference chapter for a complete description of the actions invoked by each command. As examples are presented, you'll find it instructive to have your Apple follow along.

ADDING CHART.TB TO A PROGRAM

The first step is to create a program that will serve as the framework within which we will explore the abilities of CHART.TB.

To do this we will need to follow the steps described in the beginning of this manual in adding a new command to a program.

To start, first boot on side 1 of the Trial Size Toolbox diskette, and then press ESCAPE to exit the main menu to Applesoft BASIC. Then:

1. Type in 'FP' to clear any program in memory.

2. With the Trial Size diskette (side 1) in the drive, EXEC the AMPERSAND SETUP file to create the first line of our new program. Then BRUN the Workbench and add the Toolbox File CHART.TB to the program, giving it the name CHART.

3. Then, exit the the Workbench and return to your program. Your graphics commands are now added!

4. Finally, type in: BRUN REMOVE WORKBENCH

   This will remove the Workbench utility from the computer to give us room to use the Hi-Res display.

THE WORKBENCH MUST ALWAYS BE REMOVED BEFORE RUNNING ANY PROGRAM THAT USES THE HI-RES SCREENS!

PRELIMINARY COMMANDS

Now that the graphics commands have been added to our program, we need to write the BASIC lines to use them.

Before starting the CHART.TB commands though, let's pause for a quick note about using Hi-Res graphics in Applesoft. As you may know, the Apple uses a part of memory for the Hi-Res display that actually occupies the middle of the main work area for Applesoft programs. You may have run into a resultant problem when writing very large Applesoft programs or programs that use a lot of variables or arrays. The usual symptom is some sort of strange disaster in your program when you execute an HGR or HGR2 command.

The Chart 'n Graph Toolbox actually has a number of different commands for avoiding these problems, including a program splitter that automatically protects your Applesoft program and variables from conflicts with the Hi-Res display.

In the case of CHART.TB, however, it is limited to Hi-Res page 2, so no special handling is needed for the Applesoft program itself, as long as it stays less than 56 sectors in size (disk file size).

However, variables may still present a problem, so you should always put a statement at the beginning of any program using Hi-Res page 2 to keep variables out of the Hi-Res display. This statement is LOMEM: 24576 and in all of your Hi-Res Toolbox programs, can be put on line 2 like this:

```
2  LOMEM: 24576
```

Now to actually use CHART.TB. This is done by using the "CHART" command combined with other specific commands. Let's see how this is done.

The first item of business is to set up and clear the Hi-Res page. This is accomplished with the GGO (Graphics GO) command. This is very similar in function, but not identical, to the Applesoft HGR and HGR2 commands. CHART.TB is limited to page 2

only, but the complete Chart 'n Graph Toolbox lets you use either Hi-Res page.

Thus the next line in your program should be:

```
10  &"CHART",GGO(2)
```

GGO initializes CHART.TB and displays page 2 of Hi-Res graphics. This usually is done only once, somewhere near the beginning of your program. You'll notice that CHART.TB commands are all preceded by the general name "CHART" that we gave to the CHART.TB file. This is the format that is used to execute a CHART.TB command when it is in your program with the Toolbox system.

Type in RUN now and see what happens.

The screen should clear to black as the High Resolution graphics screen is initialized.

If you type something now, you'll notice that you can't see what you are typing. This is because the graphics display (as opposed to the text display) is active.

To return to the text screen, press RESET, and add this line to your program:

```
20 &"CHART",TMD
```

Now type RUN again and see what happens.

This time, the program went to the graphics mode, and then immediately went back to the text mode. TMD stands for 'Text MoDe', and is used to switch the display back to text.

Although TMD is similar to the TEXT command in Applesoft, TMD will not reset the scrolling window parameters of the text window, just in case you modified them for some fancy text presentation.

The GMD (Graphics MoDe) does just the opposite of TMD. With these two commands you can switch back and forth between graphics and text mode. Note that neither command will affect the contents of the Hi-Res or text pages.

To see another CHART.TB command, change line 20 so that it reads:

```
20 &"CHART",RVS
```

This is the RVS (ReVerSe) command. Its function is to reverse the color of the current screen. Since you started with a black screen, the result is a white screen. If you added a line 25 with the same statement again, the screen would return to black again. At first this might seem to be a somewhat frivolous command, but later when you have some plotted data on the screen, you'll find it very useful for highlighting.

COORDINATE SYSTEMS

Before we introduce any more commands, it's probably worthwhile to briefly review the way the Apple handles the coordinate system for its Hi-Res screen, because CHART.TB handles it a little differently. The X-axis has a resolution of 280 points numbered 0-279 from left to right. The Y-axis has a resolution of 192 points numbered 0-191, and, unfortunately, they are numbered from top to bottom.

This puts the origin (0,0), somewhat awkwardly, at the top left of the Hi-Res screen. In CHART.TB, the deviation from Apple's convention is that the Y-axis has been renumbered, still 0-191, but it's numbered from bottom to top. This puts the origin (0,0) at the bottom left...where it should be.

MOVE, PLOT, DRAW AND CLIPPING

CHART.TB contains many different commands, some of which are used more often than others. We will now introduce the real workhorses of CHART.TB: PLT (PLoT), DRW (DRaW), and MOV (MOVe) . Those commands actually put the dots and lines on the screen. Each of these three commands requires that you supply an X,Y coordinate pair. PLT (PLoT) will place a single dot (in the current HCOLOR) at your specified X,Y coordinate. DRW (DRaW) will draw a line (in the current HCOLOR) to the specified coordinate from the last PLT'd, DRW'n, or MOV'd point.

In the case of the MOV command, nothing is actually drawn on the screen, but an invisible cursor is placed at the selected X,Y coordinate. This command is usually used to establish a starting point for a subsequent DRW command. OK, I can hear it now: "Big deal, you can do all that with Applesoft's HPLOT". You're absolutely right, so for our example we'll do something HPLOT can't do. Do you remember what happens when you try to specify a coordinate outside the normal Hi-Res screen, i.e. either < 0 or >191 (>279 for the X axis). Go ahead and try it. You'll find that Applesoft is very adamant about the range of coordinates it will accept. When you're done, re-write, then RUN the sample program to try the following:

```
 1 CALL PEEK(175) + 256 * PEEK(176) - 46
 2 LOMEM: 24576
10 &"CHART",GGO(2)
20 &"CHART",MOV(10,-400)
30 &"CHART",DRW(140,800)
40 &"CHART",DRW(270,-400)
```

What did we do? The first graphics command, GGO, initialized everything just like before. The second command, MOV(10,-400), moved the invisible plotting cursor to somewhere near the floor, under the left-hand side of your viewing screen. The third command, DRW(140,800), drew a line from the floor to a point near the ceiling, directly above the center of your screen.

The final command, DRW(270,-400), drew another line from the ceiling to the floor, beneath the right side of your screen. Obviously, you are going to see only those portions of the lines

that actually overlap your viewing screen. Thus in a very real sense, CHART.TB allows you to go beyond the spatial confines of your screen, and plot to an area many times larger than your screen.

Actually your screen should be considered a window through which you can view your work on this much larger, somewhat imaginary, plotting surface. That's in itself quite powerful, and not easily accomplished in Applesoft. As we go on, we'll really start to extend this plotting capability.

SCALING AND USER COORDINATES

Have you ever tried, without the advantages of CHART.TB, to plot some data to the screen? If your data was in an array, you probably had to search the array for the largest and smallest values, then you calculated some conversion factor so the converted data would range from 0-191 (Y) and 0-279 (X), thus staying within the limits of the Hi-Res screen. Or suppose you thought you knew what the scale factor would be, and along comes a data point which was a bit bigger or more negative then you anticipated... (BEEP) ILLEGAL QUANTITY ERROR. Look familiar?! Well no more! CHART.TB completely eliminates these annoying scaling problems.

You no longer have to bother about adjusting your data so it ranges from 0-191 or 0-279, because CHART.TB allows you to determine the range of values to be displayed on the Hi-Res screen. This is done with the SCL (SCaLe) command, with which you simply specify the values that the edges of your viewing window are to assume, and then proceed to PLT, DRW, and/or MOV your data.

Remember that what you are doing with the SCL command is actually redefining the coordinate system of the Apple screen (which has a maximum range of 0-191 and 0-279 without CHART.TB), to your coordinate system, which has whatever range you want. In order to make it easier to distinguish between X,Y coordinates in the two systems, throughout the rest of this manual, the Apple's screen coordinates are called screen coordinates, and your coordinates are called user coordinates ...clever!?

As an example, suppose you wanted to plot the function SIN(X) from X= -3pi to +3pi. The Y values of this function range from +1 to -1. The procedure for setting up the SCL factor can be demonstrated by entering and running the following program.

It's possible to do this in the immediate mode, but it'll be much easier to make changes if the program is entered in the deferred mode. Don't bother with the REM statements, they're just there to document our intent. Also be sure to delete any lines from the previous program except lines 1 and 2.

```
  1 CALL PEEK(175) + 256 * PEEK(176) - 46
  2 LOMEM: 24576
 10 ONERR GOTO 400 :REM Just in case of a mistake
100 &"CHART",GGO(2): REM Reinitalize everything, so
    we all start at the same place.
```

```
140 PI = 3.1415927:  REM Intialize the variable
    PI, to save time later
160 &"CHART",SCL(-3*PI,+3*PI,-1,+1): REM See below.
```

What's been done in line 160 with the SCL command is to define the left-hand side of the viewing window to be -3pi (-9.42478), the right-hand side is +3pi (+9.42478), the bottom edge is -1 and the top edge is +1. It's that simple to redefine the screen's coordinate system. Now to generate the data and plot it.

```
180 FOR X = -3*PI TO 3*PI STEP .15: REM
    FOR/NEXT Loop to generate data
200 &"CHART",PLT(X,SIN(X)): REM CHART.TB do your stuff
220 NEXT X
300 END
400 &"CHART",TMD :REM Display Text page if there's an error
```

Now run the program. You should get a wavy line made up of a series of dots. You can see that you are plotting data in line 200 in units that you have defined, the independent variable X varies from -3pi to +3pi, and the dependent variable Y ranges from +1 to -1. Once the SCL factor was set, CHART.TB handles the rest. OK, now for some changes. Instead of a point plot, let's try a line plot. Make the following additions and changes.

```
170 &"CHART",MOV (-3*PI,SIN(-3*PI))
200 &"CHART",DRW (X,SIN(X))
```

Line 200 will now draw a line between the sequentially calculated points. The additional statement, line 170, establishes the starting point for the first DRW. It's not absolutely necessary, but without it, it's hard to say where the line will be drawn from when the first DRW command is executed in the FOR/NEXT loop. Try running the modified program first with statement 170, then remove it. See!

CLEARS AND FRAME

After you run this simple program, and thought about what's been done and the speed at which it has been done, you'll have to agree that without CHART.TB, this would be somewhat tedious and considerably slower with standard Applesoft. On your own, experiment with different window sizes, ie. squares, rectangles, overlapping windows, etc, along with the RVS command.

You should also work with two other commands, FCL (Full CLear), and FRM (FRaMe). FCL clears the entire Hi-Res screen, and FRM draws a border around the screen in the current HCOLOR. This is useful for creating a border.

Before we go on, it should be mentioned, that the GGO command initializes HCOLOR=3, if you change HCOLOR, some of your PLT'ed points might not show up, and some vertically DRW'n lines might not appear. For now stick with HCOLOR=3.

There's one other point that should be mentioned regarding the variables initialized by GGO. Near the beginning of this tutorial, we did some simple plotting without having to define a

SCL'ing factor. The reason this didn't have to be specified is because the GGO command defaults the scale factor to mimic the standard Applesoft format. If you ever needed to get back to the standard Applesoft format (heaven forbid), you could do so with the following:

```
&"CHART",SCL (0,279,191,0)
```

If you can't think of a little exercise right now that uses these commands, run some of the demonstrations that have been included on both the front and back of the Trial Size diskette.

AXES GENERATION

Now it's time for some 'tinsel and flash' in the data presentation. What's a plot without some type of axes?! CHART.TB provides you with the ability to generate the usual X-Y axes within the clipping window. The command that handles the drawing of axes is AXS (AXeS). To use AXS, you must specify four variables. The first two are the separation, in user coordinates, between the tic marks on the X and Y axes. The second two variables are the X,Y coordinate where the two axes are to cross. Using the program you were working on earlier, add the following statement:

```
165 &"CHART",AXS (PI, 0.2, 0, 0)
```

Run the program. The first variable, PI, specifies a tic mark every PI unit along the X axis: there should 6 tics altogether along the X-axis (remember X goes from -3pi to +3pi). Similarly the Y-axis goes from -1 to +1, and the second variable in the AXS command, 0.2, designates a tic mark every 0.2 units along the Y-axis, therefore there should be 10 tic marks. The last two variables were set to 0,0. This designates the origin (the crossing point of the axes) at X=0 and Y=0, which in this particular example is in the center of the clipping window. If you'd like your axes to start at the lower-left, try this:

```
165 &"CHART",AXS (PI, 0.2, -3*PI, -1)
```

CONCLUSION

We hope this short tutorial will give you an idea of what CHART.TB can do for you and your programs. Following is a reference listing of all the commands included in CHART.TB.

Contrary to most Toolbox extensions, adding CHART.TB to a program automatically adds all 12 commands in one step. Ordinarily, with the Toolbox, you add just the commands you need one at a time. However, because the graphics commands are so interelated, there is no great space savings in deleting any given command from the group, and most of the time the complete set is used in any given program.

Remember that the Chart 'n Graph Toolbox has a total of 27 actual graphics commands in its equivalent Toolbox File to CHART.TB, plus many additional commands for Hi-Res graphics programs.

>> CHART.TB <<

CHART.TB is a library of 12 new graphics related Applesoft commands. They can be used on any 48K Apple II with ROM Applesoft or the Language System, and one disk drive (DOS 3.3).

Each of the 12 commands performs a single function which give the programmer the building blocks needed to design his own plotting package, as simple or complex as is needed.

The names of the new commands are:

GGO	TMD	GMD	SCL	FCL	RVS
MOV	PLT	DRW	BIT	AXS	FRM

Each command will now be discussed in detail. Examples follow the descriptions.

LENGTH: 3152 ($C50) bytes

SYNTAX: &"NAME",command (variable list)

SAMPLE:
```
&"CHART",GGO(2)
&"CHART",AXES (XI,YI,0,0)
&"CHART",AXES (MX/10,MY/10,OX,OY)
```

Command: GGO(page)

Meaning: Graphics GO

Purpose: Initialize graphics (similar to HGR & HGR2)

Variables: page - The number (Trial Size limited to '2') of the Hi-Res graphics page to be used.

Results:
The selected Hi-Res screen is displayed.
Full screen graphics is set (no text at bottom).
The screen is cleared to Black1.
HCOLOR is set to 3.
The coordinate (0,0) is redefined to be in the lower left hand corner of the screen.
The clipping window is set for full screen use (0-279,0-191).
The screen is scaled in X from 0-279 and in Y from 0-191.
Both axes are set to linear mode.
The X-axis tic size is set to 2 dots and the Y-axis tic size is set to 3 dots.

Example:
```
10 &GGO(1)
```

Command: TMD

Meaning: Text MoDe

Purpose: Display text screen (similar to TEXT)

Variables: none

Results: The primary text page is displayed.
The scrolling parameters of the text window are not reset (unlike TEXT).

Example: 20 &TMD

Command: GMD

Meaning: Graphics MoDe

Purpose: Display last used Hi-Res graphics screen

Variables: none

Results: The last used (either primary or secondary) Hi-Res graphics page is displayed.

Example: 30 &GMD

Command: SCL(uxmin,uxmax,uymin,uymax)

Meaning: SCaLe

Purpose: Define the screen's range in user coordinates

Variables: uxmin - user's defined value for the left side of the current clipping window.
uxmax - user's defined value for the right side of the current clipping window.
uymin - user's defined value for the bottom of the current clipping window.
uymax - user's defined value for the top of the current clipping window.

NOTES: Uxmin must not equal uxmax
Uymin must not equal uymax

Results: All move, plot, draw and label commands are made in these new user defined coordinates.
This scaling remains in effect until another SCL is issued or graphics is re-initialized with a GGO.

Remarks: Reverse scaling can be defined by setting uxmin less than uxmax and/or uymin less less than uymax.
Scaling defaults to linear, but can be made logarithmic by the LMD command.

Example: 90 &SCL(-100,100,-50,150)

Command: FCL

Meaning: Full CLear
Purpose: Clears full screen

Variables: none

Results: Clears whole screen to Black1.
All scaling parameters remain unchanged.

Remarks: This is a fast clear - faster than HGR used by Applesoft.
Useful for animation.

Example: 120 &FCL

Command: RVS

Meaning: ReVerSe

Purpose: Reverse everything in the current clipping window

Variables: none

Results: All colors are reversed in the current clipping window (bit 7 is unchanged).
Everything else on the screen is unchanged.

Remarks: Can be used to simulate a blinking cursor by alternatively reversing a given area.
Also useful for highlighting titles.

Example: 130 &RVS

Command: MOV(ux,uy)

Meaning: MOVe

Purpose: Move to a user specified point without line generation

Variables: ux - user X coordinate
uy - user Y coordinate

Results: Establishes a starting point for the next line to be drawn.
Display remains unchanged.

Example: 140 &MOV(-50,-35.7)

Command: PLT(ux,uy)

Meaning: PLoT

Purpose: Plot a point at the coordinate specified by the user

Variables: ux - user X coordinate
uy - user Y coordinate

Results: Leaves a point, in the current HCOLOR, at the user scaled coordinate.
Will not be displayed if outside clipping window.
Establishes starting point for line generation.

Example: 150 &PLT(79,140)

Command: DRW(ux,uy)

Meaning: DRaW

Purpose: Draws a line to user specified coordinate

Variables: ux - user X coordinate
uy - user Y coordinate

Results: A line is drawn from the last point moved to, drawn to, or plotted at, to the user scaled coordinate.
Line is drawn in current HCOLOR.
Any parts of the line falling outside the clipping window are not displayed.

Example: 160 &DRW(200,300)

Command: BIT(sx,sy,color)

Meaning: BIT color

Purpose: Determine color of bit at screen location sx,sy

Variables: sx - screen X coordinate of bit (0-279)
sy - screen Y coordinate of bit (0-191)
color - color of bit (must be a variable)

Results: Color of bit at sx,sy is returned. Value will be one of (0,1,2,4,5,6).
Note: white1 and white2 cannot be determined.

Remarks: Can be used to write custom screen dump routine.

Example: 170 &BIT(12,118,C)

Command: AXS(xtic,ytic,xorigin,yorigin)

Meaning: AXeS

Purpose: Draw the X-Y axes in clipping window

Variables: xtic - distance between X-axis tic marks in user

coordinates.
ytic - distance between Y-axis tic marks in user coordinates.
xorigin - X coordinate of Y-axis.
yorigin - Y coordinate of X-axis.

Results: The X-Y axes is drawn in the current clipping window in the current HCOLOR.
When in log mode, the xtic and/or ytic parameter(s) specify that the tic marks are to be placed at multiples of 10 divided by the tic mark parameter. Thus, if LMD(1) is in effect and xtic=10 is specified, there will be nine tic marks in each decade located at multiples of (1,2,3,4,5,6, 7,8,9). If xtic=5 is specified there will be five tic marks per decade located at multiples of (1,2,4,6,8). xtic=2 will specify tic marks at multiples of (1,5). The four log tic mark parameter values (1,2,5,10) are generally the most useful since they are the only values of the parameter that specify integer tic mark locations. When in log mode, the xorigin and yorigin parameters are the same as previously defined. Resets starting point for DRW.

Remarks: If HCOLOR=1,2,4, or 5, some lines may not be displayed (adjust clipping window or use White1 or White2). Close tic marks may not appear evenly spaced due to the limited screen resolution.

Example: 180 &AXS(10,10,0,0)

Command: FRM

Meaning: FRaMe

Purpose: Outline current clipping window

Variables: none

Results: The current clipping window is outlined in the current HCOLOR.

Remarks: Depending on HCOLOR, some vertical lines may not be visible (adjust window). Frame is just inside window.

Example: 250 &FRM

>> FIXED LENGTH INPUT.TB <<

by Steve Cochard

FUNCTION: This command will allow the programmer to specify the allowable length of an input string. In addition it will allow a prompt string to be printed, print a number of underscores equal to the allowable length of the string, allow default strings to be specified, and posseses some very nice extra features.

LENGTH: 490 bytes ($1EA)

SYNTAX:
```
& "NAME", [ slit;] aexp, svar [,aexp] [,svar]
& "NAME", [ prompt;] len of input, input str$
          [, GOTO line #] [,Control Char]
```

SAMPLE:
```
&"FIXED LENGTH INPUT","ENTER A STRING";L,A$
&"FIXED LENGTH INPUT",L(1)* VAL (A$(I%)),CD$(X
   + L(2))
&"FIXED LENGTH INPUT",15,A$,GT,CTRL$
```

If the input string (svar) is a null string then no default will be printed, if it is not then the current value of the input string will be printed as the default. If RETURN is pressed as the only input the default string will be returned as the user's input. If the user changes his mind and back spaces to the start of the input line and THEN presses RETURN, the default will again be returned as the user's desired input.

Note the additional, optional variables in the above syntax and examples. If a string variable (svar) is specified, it will "activate" the control keys during the first character of input. If the first character is a control character then it is returned in the specified string variable. The default string will remain unchanged and the input will be terminated.

If a numeric expression (aexp) is specified, the ESC and left arrow keys are "activated" during the first character of input. If ESC is pressed as the first character of input, the routine will execute a GOTO statement, where the numeric expression contains the line number to GOTO.

HOW TO USE: This command is very useful for almost all of the input a program would require. The fact that the allowable length of the input is shown on the screen as well as the default string features add a high degree of professionalism to any program.

The default string feature allows the program(mer) to select what would be the most likely input and display it. If the default input suits the user he only needs to press return and the default is accepted. If the default is not acceptable to the user he simply types in his response to the input request. The following example will illustrate the command in action:

Command executed:

```
90 A$=""
100 &"FIXED LENGTH INPUT","USER ID:  ";5,A$
```

Screen display:

USER ID: _____

User responds with "4123" and the screen displays:

USER ID: 4123_

If a default value is desired, the input string needs only to be assigned the default value in advance. for example:

```
100 A$="CAR PAYMENT"
110 &"FIXED LENGTH INPUT","CHECK WRITTEN TO:  ",11,A$
```

When these commands are executed, the screen will respond with the following:

CHECK WRITTEN TO: CAR PAYMENT

If that is the correct response, the user simply presses RETURN and CAR PAYMENT is accepted as the value of A$; if it is not the correct response, the user simply types in the correct one.

If the response was to be "JOES BAR", the screen would look like this:

CHECK WRITTEN TO: JOES BAR___

One other point to remember: if the user begins to type a response and he decides that the default was the correct response after he typed something in, all that needs to be done is for him to backspace to the first character, and press RETURN, and the default will be accepted as the user's response.

>> USE OF THE "SPECIAL FEATURES" <<

If a string variable is added to the command's variable list like this:

```
&"FIXED LENGTH INPUT",10,A$,C$
```

then it is possible to have any control character pressed by the user as the first character of input assigned to the additional string variable. The default string will not be changed. Note that the entry of a control character as the first input will also automatically generate a RETURN character and terminate the user's input.

This feature is useful when various options are available to the user at different "levels" of the program. Many database programs use control characters to get commands from the user to add or delete records, terminate the input session, etc. With this command it is very simple to do the same thing within your programs without the extra BASIC programming overhead!

If an arithmetic expression (number, variable or expression) is added to the command's variable list, the ESC and left arrow

keys are activated. The complete statement would then look like this:

```
&"FIXED LENGTH INPUT",10,A$,J
```

With this statement, "J" contains the value for the statement to GOTO if the user presses the ESC or left arrow key as his first character of input. As with the optional string variable described earlier, the command will automatically provide a RETURN, terminate the input, and then perform the GOTO.

This is perhaps the most powerful feature of FIXED LENGTH INPUT. The FIXED LENGTH INPUT DEMO gives some very nice examples and ideas for using this option.

Note that it is possible to add both of the optional variables to the command's variable list to activate BOTH the ESC/left arrow key and control character return. This will provide the programmer with the ultimate in this command's flexibility.

If during the normal input, any control character is input, then a beep will sound, notifying the user that a control character is not acceptable. This prevents unwanted control characters from unknowingly getting into the input strings.

ERRORS: TYPE MISMATCH ERROR will occur if the input string is not a string variable.

OTHER ERRORS: Any error which may be encountered when evaluating expressions (during evaluation of the allowable length). Also, BAD SUBSCRIPT error if the input variable is an array which has a bad subscript, SYNTAX ERROR if the syntax of the command is incorrect. UNDEF'D STATEMENT ERROR will occur if the optional numeric variable is used and the referenced line number does not point to an existing program line.

>> GET.TB <<

by Peter Meyer

FUNCTION: Gets a character from the keyboard, checks to see if it is a character you want, and returns it in a string variable, with or without screen echo and carriage return.

LENGTH: 255 ($FF) bytes.

SYNTAX:
```
& "GET" A$
& "GET" A$ IF B$
& "GET" A$ IF B$, FL
```

In each of these cases:

(1) A prompt string may be included, as in:

```
& "GET", "PRESS KEY "; A$
```

(2) The second variable (if present) may be a string literal, rather than a string variable, as in:

```
& "GET",A$ IF "YN"
```

The second variable may in fact be any string formula, whether a simple or array variable such as X$(I), a string literal or a formula such as:

```
MID$(X$(I),VAL(P$),VAL(L$)) + CHR$(27)
```

(3) A semi-colon may be appended, as in:

```
& "GET", "PRESS A KEY "; A$ IF B$, FL;
```

USE:

The command statement:

```
& "GET" A$;
```

is equivalent to the Applesoft command: GET A$.

GET.TB goes beyond GET in the following ways:

(1) You may include a prompt string, as with the standard Applesoft INPUT statement. In this respect, GET.TB resembles the Applesoft INPUT statement rather than the GET statement.

(2) You may require a carriage return to be sent automatically. If a semi-colon is not present at the end of the variable list then a carriage return will be sent, otherwise the carriage return is suppressed.

(3) You may check the key pressed by the user so as to admit the character input only if it occurs in a specified character string. This 'screenstring' (which is used to screen out undesirable characters) is specified by means of the second variable, and may be a string variable (e.g. AI$(I)) or a string

literal (e.g. "YN"). The screen-string may contain any character, including control characters. For example, if we define the second variable by means of:

```
B$ = CHR$(1) + CHR$(2) + CHR$(3)
```

then the command statement:

```
& "GET" A$ IF B$;
```

will work as does the Applesoft command GET A$, except that only control-A, control-B and control-C can be got.

NOTE: Some utility programs such as G.P.L.E., and in fact the Apple //e and //c input routines screen keyboard input themselves, before passing it on to whoever or whatever (DOS, Applesoft, your command) is next in line. If one of these is in effect, then control characters, including the ESCAPE character, may disappear mysteriously after input from the keyboard, before GET.TB ever sees them.

If an empty screen-string is specified, e.g.

```
& "GET" A$ IF "", 1
```

then the input is not screened; thus all characters are acceptable.

If the screen-string is not empty and the character got does not occur in it, then GET.TB will beep discreetly and await a new keypress. Thus it will wait until an acceptable key has been pressed before returning a value in the input variable.

(4) You may allow the user to accept a default value simply by pressing RETURN (see below). The default value is always the first character in the screen-string.

(5) You may specify that the character got, if acceptable, should be echoed to the screen (see below). Or you can suppress the screen echo, if you wish.

The options allowed under (4) and (5) are specified by calling GET.TB with the third variable (FL in the examples above) set to a certain value. The allowable values, and their meanings, are as follows:

Value	Effect
0	No default allowed, no screen echo.
1	No default, but screen echo.
2	Default allowed, but no screen echo.
3	Default and screen echo.

The default value for the third variable is 0.

If default is in effect then the default character will be flashed at the current cursor position, unless it is a control character. This informs the user of the default value.

If echo is in effect then the character got is echoed to the screen (at the current cursor position) unless it is a control character.

GET.TB uses the RDKEY subroutine at $FD0C. Thus it gets data from whatever is the current input device. This is usually the keyboard, but if a DOS READ command is in effect then it is the disk.

LIMITATIONS:

Clears keyboard buffer before checking so keyboard buffers may not work as expected. This is usually an advantage in that accidental keypresses before a given menu will not trigger a menu choice, as is the case with the usual Applesoft GET.

ERROR MESSAGES:

SYNTAX ERROR if a comma is used instead of a semicolon after a prompt string.

OUT OF DATA if default allowed but the screen-string specified is empty (so that no default character is defined).

ILLEGAL QUANTITY if third variable is other than 0, 1, 2 or 3.

SAMPLE PROGRAM:

This program asks the user if a catalog is needed.

```
10  CALL PEEK(175) + PEEK(176)*256 - 46
20  & "GET", "WANT CATALOG?  "; A$ IF "YN", 3
30  REM  Default = "Y" and screen echo required.
40  IF A$ = "N" THEN PRINT: GOTO 20
50  PRINT CHR$(4)"CATALOG"
```

Because DOS commands do not work immediately after a normal Applesoft GET command, GET.TB has been designed so as to avoid this by printing a carriage return (unless this is suppressed by a semi-colon), which fixes things up for DOS. The following program shows what can happen with the standard Applesoft GET command:

```
10  PRINT "WANT CATALOG?  ";: GET A$
20  IF A$ = "N" THEN PRINT:  GOTO 10
30  PRINT CHR$(4)"CATALOG"
40  REM 'CATALOG' DOESN'T WORK
```

DEMO PROGRAM:

Run the program GET DEMO to see how GET.TB works in practice. This program asks you to define a screenstring. Any keyboard character may be part of the screen-string, including control characters.

>> PTR READ.TB <<

by Roger Wagner and Craig Peterson

FUNCTION: This will read any two byte pointer in memory, and return the decimal value in a numeric variable.

LENGTH: 156 bytes ($9C)

SYNTAX: &"NAME",address,variable
&"NAME",aexpr|hexnum,avar

SAMPLE: &"PTRD",175,N
&"PTRD",AD,X
&"PTRD",$73,N

HOW TO USE IT: This is most useful when trying to determine the contents of the many two byte pointers that Applesoft uses to keep track of its own memory usage. To determine where one of these byte pairs is currently pointing, follow the command name with the address of the first byte in the pair.

The address may be specified in either decimal or hex notations. If decimal is chosen, follow the command name with a numeric constant, variable or expression.

If the address is to be specified in hex, simply use the hex value, beginning with a dollar sign ($) and followed by one to four hex digits.

After the specified address, place a real or integer variable into which the returned address is to be placed. Real variables will return with values in the range of 0 to 65535. If an integer variable is used, values over 32767 will be expressed in their negative forms. If a hex address is desired, XNUM.TB may be used to convert decimal addresses to hex if desired.

LIMITATIONS: If hex addressing is used, the address must be specified within a string. All addresses must be in the range of -65535 to 65535 or $0 to $FFFF.

SAMPLE LISTING:

```
 1 CALL PEEK(175) + 256 * PEEK(176) - 46
10 &"PTRD",$AF,N
20 PRINT"END OF PROGRAM: ";N
30 &"XNUM",N,H$
40 PRINT"(HEX:            ";H$;")"
```

>> TEXT OUTPUT.TB <<

by Steve Cochard

FUNCTION: Prints text to screen or printer without word breaks at right margin.

LENGTH: 286 bytes ($11E)

SYNTAX:
```
&"NAME",sexpr [,aexpr ] [,aexpr ] [ AT aexpr ] [;]
&"NAME",string [,width ] [,indent ] [ AT spacing ] [;]
```

SAMPLE:
```
&"TEXT","THIS IS A TEST"
&"TEXT",A$;
&"TEXT",B$ AT 2
&"TEXT",A$,75,5 AT 3;
&"TEXT",A$+B$,80 AT 2
```

HOW TO USE IT: This routine is similar to the Applesoft PRINT statement, except that it is designed to eliminate the splitting of words at the ends of lines.

As in the standard PRINT statement, the presence of a semicolon at the end signifies that no carriage return is to be printed after the final character.

If the width variable is not specified then it is assumed to be the current screen width. Thus you can omit this variable when outputting to the screen with the text window set normally or using the WINDOW.TB command. When sending text to a printer, the width should be set to the maximum number of columns supported (usually 80).

When text is being displayed on the screen, you may indent the first word (as at the start of a paragraph) by using the HTAB command to position the cursor appropriately before calling the routine. When printing multiple sentences, it is recommended that you define each string with two spaces following the period at the end. This will then properly format entire paragraphs to the screen.

Each variable in the variable list after the string to be printed is optional. This means you can omit any of the values you wish and the current screen values / print defaults will be used. The defaults are:

column width:	Uses current text screen right window boundary. Automatically adjusts for an indent value or left window setting if needed.
indentation:	Uses current text screen left window boundary.
spacing:	Uses a value of '1' (i.e. normal single spacing).

If the semi-colon (";") is not used, a carriage return will be printed at the end of the output string.

Using the WINDOW.TB command is the preferred way of determining the left are right boundaries for your text. The use of specific

variables is allowed to make this command usable for printer output.

LIMITATIONS: The text printed must be a string; numeric data is not allowed. You may however use Applesoft's STR$() function to convert any number to a string.

The width variable must be not be less than the length of the longest word in the text. Values less than '20' are dangerous in this regard. If you are specifying the allowable width yourself, remember that on a 40 column screen, an indentation value of '5' leaves a maximum width of 35 characters.

SAMPLE LISTING:

```
10  CALL PEEK(175) + PEEK(176) * 256 - 46:  HOME
20  A$(1) = "ONCE UPON A TIME THERE WAS A DARK AND DEEP
    FOREST, FILLED WITH MYSTERIOUS PRESENCES, "
30  A$(2) = "LIKE ELVES, TROLLS, PIXIES, GIANTS AND
    ORCS.  "
40  A$(3) = "ONE DAY BO-PEEP CAME ALONG, LOOKING FOR
    HER SHEEP ... "
40  J = 1
50  HTAB J
60  FOR I = 1 TO 3
70  & "TEXT", A$(I);
80  NEXT: PRINT: PRINT
90  J = J+5:IF J < 30 THEN 50
```

>> WINDOW.TB <<

by Joe Holt

FUNCTION: It does windows, of course! WINDOW.TB provides a POKEless (as well as paneless) way to set the top margin of the text window (or lo-res graphics screen, although this has absolutely no use) and optionally set the top, left, and right margins.

LENGTH: 95 bytes ($5F)

```
SYNTAX: & "NAME",top margin [,bottom margin [,left
           marg in [,right margin]]]
        & "NAME",aexpr [,aexpr [,aexpr [,aexpr]]]

SAMPLE: & "WINDOW",1,24,1,40          (Kinda like TEXT)
        & "WINDOW",X,X+5
        & "WINDOW",T,B,L,R
        & "WINDOW",T*COS(0),T*2,L,SQR(R)^.5
```

HOW TO USE IT: This command allows the user to set the text window by specifying the top margin and optionally specifying the top and left margins and even more optionally the right margin. Note that, unlike the standard Applesoft POKEs to set the window, you should specify the window's right MARGIN, NOT the window WIDTH (like we should've been able to do to begin with). If the optional variables are not specified, their corresponding margins will not be affected.

The margins specified are to be the first character positions within the text window. For example, an &"WINDOW",5,20,5,36 will leave a protected border around the text window four characters (or lines) wide (or high).

All variables may be integer or floating point expressions or variables. The top and bottom margins must be in the range 1..24 (that's one through twenty-four) and the left and right margins must be in the range 1..80, although a value greater than 40 while using the standard Apple 40 column screen should be accompanied by a prayer and sacrifice.

If you attempt to use the command with a bottom margin higher than the top, or a right margin less than the left (or a top margin lower than the bottom, or a left margin greater than the right), an ILLEGAL QUANTITY ERROR will result, so don't even try it.

LIMITATIONS: This command sets the text window only, it does not position the cursor within the window, nor does it clean the window.

SAMPLE LISTING:

```
  1 CALL PEEK(175) + 256 * PEEK(176) - 46
 10 TEXT : HOME
 20 PRINT "You are now entering....The Window Zone!";
 30 PRINT "+--------------------------------------+"
 40 FOR V = 3 TO 23
 50 VTAB V :HTAB 1 :PRINT "!"; :HTAB 40 :PRINT "!";
 60 NEXT V
 70 PRINT "+--------------------------------------";
 80 POKE 2039, ASC("+") + 128 :REM MUST DO TO KEEP
    FROM SCROLLING
 90 &"WINDOW",3,23,2,39
100 HOME : END
```

THE TOOLBOX SERIES

Following is a list of all the new commands included on each Toolbox Series package. Each Toolbox package comes with a copy of the complete Workbench to add the commands of your choice.

THE DATABASE TOOLBOX

This Toolbox Series library is dedicated to doing just about everything you ever thought possible (and some impossible) to an Applesoft array. All routines work with two-dimensional arrays, many times with both string and data. Routines to perform virtually any usual mathematical operation on an entire array in one statement.

ARRAY READER: Reads a sequential text file from disk into either a string or real (floating point) array. Also evaluates expressions being read from a text file into a real array.

ARRAY WRITE: Writes the contents of any string or real (no "%" sign in the variable name) array to a specified sequential text file at extremely high speed. The range of data elements within the array may be specified as well as the size and location of the data buffer used by this subroutine.

ARRAY SEARCH: Searches a oneor two-dimensional string array for a specified string, sub-string, or Boolean relationship between the search string and the found string.

ARRAY SORT: Quickly sorts a oneor two-dimensional string, real or integer array using the Shell-Metzner method and the use of multiple key fields in either ascending or descending order.

DELIMITED STRINGS: A package of interrelated routines that allow a one-dimensional string array to be manipulated in a record/field format similar to what can be done with a two-dimensional array, including find a field, replace a field, search the entire array for a specified string and sort either part of or the entire array on any field.

ARRAY RENAME: This routine will rename an existing array.

ARRAY CLEAR: A time saving device that that clears a specified array almost instantly, without deleting the array from memory, but rather, "zeroing" the elements within it.

ARRARY DELETE: Completely deletes an existing array from memory.

ARRAY REDIM: Allows a previously dimensioned array to be redimensioned without any loss of data. Works with integer, string and real arrays. It will redimension oneand two-dimensional arrays only.

ARRAY ROW COL ADD: Adds any specified row or column in a two-dimensional real, integer or string array.

ARRAY ROW COL KILL: Kills any specified row or column in a two-dimensional real, integer or string array.

ARRAY FUNCTIONS, ARRAY FUNCTIONS SINGLE , ARRAY RANDOM, and ARRAY FUNCTIONS PARSE: A collection of routines that allow any one of over 25 Applesoft functions to be performed on every element of an array at once. In addition, the ARRAY FUNCTIONS PARSE routine allows complete expressions to be performed on entire arrays.

ARRAY STR$: Converts a real (floating point) array into a string array.

ARRAY VAL: This routine attempts to interpret an entire string array as real numbers and return those values to the specified real array.

ARRAY ROUNDING: Takes any real array and rounds every element within it to the number of specified decimal places.

MATRIX IDENTITY: Returns the identity matrix in the designated array. The identity matrix is a constant matrix wherein the diagonal consists entirely of '1's and all other positions are filled with '0's.

MATRIX TRANSPOSE: This will return the transpose of the designated array.

MATRIX MULTIPLY: Returns the matrix product of two real numeric arrays.

MATRIX INVERSION: Returns the inverse (also called the determinant) of the designated square real array.

USR MACHINE: This routine is a utility to be used in conjunction with both the Toolbox and USR compatible machine language routines, such as the USR routines in ARRAY. It allows for simple interfacing of routines to the Applesoft USR function, similar to using shape tables with the Toolbox.

USR FUNCTIONS: These routines interface with the USR function to add the functions listed below to Applesoft. They may be used with the USR MACHINE or independantly.

LOG 10 will return the logarithm to the base 10 of the given expression.

SINH will return the hyperbolic sine of he give expression.

COSH will return the hyperbolic cosine of the given expression.

SECANT returns the secant of the given expression, where the expression is given in radians.

ARC COS will return the inverse cosine of the given expression. The result is returned in radians.

ARC SIN will return the inverse sine of the given expression. The result is returned in radians.

RAD DEG will convert radians to degrees (expressions given in radians with the result given in degrees).

DEG RAD converts degrees to radians.

INVERSE TANH returns the inverse hyperbolic tangent of the given expression.

WORD PEEK will perform a two byte peek of the memory location specified by aexp.

RND FIX: Seeds the Applesoft random number generator with the correct value.

ONERR: Allows the use of the Applesoft ONERR statement to occur anywhere within an Applesoft program line.

RETURN FIX: Executes the correct code to implement an Applesoft RETURN procedure.

GARBAGE MAN: Performs the same function as FRE(0), dead string housecleaning, but up to twenty times faster.

DISK START UP: Activates the disk drive specified in the command.

FAST FP RUN: This routine will LOAD and RUN an Applesoft program at extreme speed.

FAST BRUN: This BRUNs a binary disk file extremely fast. Also allows a parameter list to be added to the syntax to facilitate machine language overlays.

THE CHART TOOLBOX

This Toolbox Series package contains a library of Hi-Res chart (plotting) routines that really helps with those custom graphics programs. Doing stock analysis, sales forecasts, scientific applications or even just plotting the monthly budget, the Chart Toolbox makes the job much easier.

AMPERCHART: A 27 command graphics extension for Applesoft, designed to greatly simplify the task of creating custom graphics programs. Provided in normal and Double Hi-Res graphics versions (Double Hi-Res requires an Apple //c or //e with 128K RAM).

ARC: Generates arbitrary sections of regular polygons which in the limiting case (as sides get large) approach arcs of circles.

AXS: Draws an X-Y axes within the current window.

BIT: Determines the color of the bit at the specified screen location.

CLP: Defines a clipping window, as a subset of the Hi-Res screen, in which all plotting commands will be confined.

CLR: Clears the current clipping window.

DRW: Draws a line from the last coordinate to the next user specified coordinate.

DSP: Displays either the primary or secondary Hi-Res page independently of which one is currently in use.

FCL: Clears the full Hi-Res screen.

FIL: Fills a black convex area with the current HCOLOR.

FRM: Frames or outlines the current clipping window.

GID: Draws an axes and grid lines in the current window.

GGO: Command used to initialize graphics.

GMD: Displays the last used Hi-Res graphics screen.

HLB: Places a horizontal label at the user specified screen location.

LMD: Defines the X and Y axes to use either Linear or Log scaling.

LOC: Returns the current location of AMPERCHART.

MIX: Displays graphics and last four lines of text.

MOV: Moves cursor to a user specified point without drawing a line.

NMX: Displays full graphics screen. Opposite of MIX.

PLT: Plots a point at the coordinate specified by the user.
RVS: Reverses everything in the current clipping window.

SCL: Defines the scale or range in user coordinates for the current clipping window.

STU: Enables conversion of the screen coordinates into user scaled coordinates and returns the values in UXV and UYV.

TMD: Selects the Text mode.

TSZ: Adjusts the tic mark sizes of the axes.

VLB: Places a vertical label at the user specified screen location.

WRK: Defines the Hi-Res page to be worked on independent of which page is currently displayed.

OTHER COMMANDS: In addition to the 27 commands added by the AMPERCHART Toolbox file itself, other individual commands are also provided in the Chart 'n Graph Toolbox:

WINDOW: Saves all or part of any normal or Double Hi-Res screen, to an Applesoft array. This makes it possible to save portions of the screen to save and later re-load from disk, and to create pull-down menus similar to the Macintosh using the Hi-Res screen.

SPLITTER: Splits any Applesoft program, and any machine language routines that may be attached to it, around the Hi-Res graphics pages.

HI/LO ZOOM: Transfers a block of graphics information from either Hi-Res page to LoRes Page 1 for easy editing.

HIRES ZOOM: Expands the size of any area on the active Hi-Res page by a factor of two.

MEDIAN FILTER: Implements a median filter over any specified area on the active Hi-Res page. Smoothes out "blockiness" associated with the Hi-Res zoom.

BLOAD PIC: BLOADs a Hi-Res picture much faster than normal DOS.

BSAVE PIC: BSAVEs a Hi-Res screen to disk.

DUMP: A set of Hi-Res graphics printer dump routines for various models of Apple ImageWriter, Epson and Anadex printers.

THREE-D ARRAY: Transforms the data points of a three-dimensional object into a two-dimensional (T.V. screen) projection.

IDENTIFICATION: Allows a program to determine what type of Apple it is running on, and whether an 80 column card and extended memory are available.

SHAPE MAKER: A utility for creating your own shape tables.

DOUBLE HI-RES ROUTINES: This is a set of Double Hi-Res drawing commands that can be used on an Apple //c or //e with 128K RAM. This is different than the AMPERCHART command set itself in that these commands are direct replacements for the usual Applesoft commands such as HPLOT, DRAW, etc. Under DOS 3.3, a modified Applesoft can be used (provided in the Chart 'n Graph Toolbox) that allows unmodified Applesoft programs to directly function in the Double Hi-Res mode.

DHR TRANSFER: This command transfers Double Hi-Res images from main to auxillary memory and back again. This provides the capability of both storing and retrieving up to 5 Double Hi-Res images in auxillary memory of the Apple //e or //c, and also a means of using DOS 3.3 to load and save Double Hi-Res images.

This is an excellent "Sampler" package if you just can't decide what Toolbox Series package to get first. This is a varied collection of valuable and multi-use commands that can be used in just about every program you write.

PRINT USING: Formats numeric data for screen, printer, or disk file output.

TEXT OUTPUT: Prints text to screen or printer without word breaks at right margin.

STRING INPUT: An alternative to the usual Applesoft INPUT statement, allowing commas and colons in input strings.

STRING SEARCH: Finds a substring within another larger string.

ARRAY SEARCH: Searches a one-dimensional string array for the occurrence of a specified search string, or for the occurrence of a string standing in a certain relation to the search string.

BUBBLE SORT: Sorts the elements of one-dimensional string array, placing the empty strings at the end.

TONE: Generates a pure tone of a given pitch and duration. Can also be used as a pause.

SOUND EFFECTS: Allows creation of unusual sound effects for program use.

SWAP: Swaps two Applesoft variables without requiring a third variable, and without generating "garbage" which would have to be handled later.

ERR: This fixes the stack pointer in preparation for continuing the operation of a running Applesoft program when RESUME will not be used. It will also optionally return the error code and line number of the error.

ERR MSSG: Prints the usual Applesoft or DOS error message from within a running program without halting program execution.

GOTO: Allows the equivalent of the GOTO statement in Applesoft with line number specified being given by a variable.

GOSUB: Allows the equivalent of the GOSUB statement in Applesoft with the line number specified being given by a variable.
LINE DATA RESTORE: Performs a similar function to the Applesoft RESTORE command, with the exception that the line number to which the DATA pointer is restored can be specified.

DATA ELEMENT SELECT: This advances the DATA pointer a given number of positions relative to its current position. This gives a random access-like aspect to DATA statements.

XNUM: This will convert numbers between decimal and hexadecimal notations.

MEMORY MOVE: Copies a block of memory from one location to another.

RESTORE AMPERSAND: Restores the ampersand vector to its original value before the Applesoft program was run.

PTR READ: Reads any two byte pointer in memory, and returns the decimal value in a numeric variable.

PTR WRITE: Sets any two byte pointer in memory to the value or address specified.

SHAPE GOBBLER: Converts existing Applesoft shape tables into usable Routine Machine modules.

SHAPE VIEWER: Allows examination of raw shape tables.

HIRES ASCII: A shape table of 95 ASCII characters used for printing text on the Hi-Res screen.

GAME SHAPES: A short shape table of the famous 'DOT' shape used to create the 'laser ray' used in the opening screens of the Trial Size Toolbox, plus an extra pair of 'explosion' shapes.

MISC. SHAPES: 60 assorted shapes including buildings, a castle, mountains and ships.

HIRES TEXT PRINTER: Uses any Applesoft shape table such as HIRES ASCII to print text (or any graphics shapes) to the Hi-Res screen.

TURTLE GRAPHICS: Provides a set of Hi-Res graphics drawing routines based on the "turtle graphics" concept.

BLOAD: BLOADs binary files approximately four times faster than normal DOS 3.3

BINADR: Returns the length and load address of any binary file on a disk, WITHOUT loading the file.

RESET ONERR: Sets the RESET vector so as to generate an Applesoft error code when RESET is pressed.

RESET RUN: Sets the RESET vector so as to re-run the current Applesoft program when RESET is pressed.

RESET BOOT: Sets the RESET vector so as to re-boot DOS when RESET is pressed.

FREE SECTOR COUNT: Allows determination during program execution of the number of free sectors on a disk in a selected drive.

PLUS!!! - The Wizard's Toolbox is also the only Toolbox package that contains extensive information on writing your own machine language Toolbox commands. If you've ever had an interest in writing machine language programs, the Toolbox system is one of the best ways to start putting machine language power to use in your own programs. It's easier than could imagine, and you can create useful routines of your own with a minimum of programming time!

THE VIDEO TOOLBOX

This Toolbox Series package is a collection of text screen related routines dedicated to text input and display.

MENU GENERATOR: If you've seen the great menus used on the HELLO programs on the Trial Size diskette and other Roger Wagner Publishing disks, you know what this command can do! One statement creates instant "file folder" menus with any number of menu choices, automatically!

INPUT USING: A special input routine to allow formatted INPUTs, checking for predefined allowable characters in the input line and providing 'protected field' input.

EDIT STRING: A string input routine with comprehensive editing capabilities.

LOWER CASE INPUT: String input routine that facilitates the entry of lower case characters, commas, colons, quotation marks, from keyboard or text files on disk.

PRINT JUSTIFY: Routine to print centered, left, right or fill justified string text.

NUMBER FORMAT: Formats numeric data for screen, printer or disk file output, or to be placed into a string variable.

SINGING KEYS: A simple utility to produce a tone from the speaker each time a key is pressed.

PRINT REPEAT: Prints multiple copies of a given string.

SPEED CONTROL: Provides keyboard control of printout speed.

CURSOR MOUSE: Permits user input from the video screen using a mouse input device, joystick, or game paddles.

SCROLLER: A specialized routine to scroll messages on the 40 column screen.

SCROLL.4X: 4 way scrolling for Text screen.

LO-RES SCROLL: 4 way scrolling for the Lo-Res graphics screen.

SHOW CTRL CHAR: Displays all control characters sent to the screen (except carriage return) as inverse letters.

SCREEN DUMP: Dumps the current 40 column or 80 column (Apple//e) screen to a printer or other output device.

MEMORY DUMP: Provides an easy way to examine internal memory in both hexadecimal and ASCII.

DISASSEMBLE MEMORY: Provides an easy way to disassemble and examine internal memory.

WAIT: Waits for the specified amount of time (up to several minutes) and then resumes program execution.

KYBD: Scans the keyboard and if a key was pressed, since the last KYBD or input, and returns the character in a string variable.

LOWER CASE: Converts all output from the computer to lower case.

GOTOYX: Positions the screen cursor to the specified X and Y coordinates and optionally prints an expression.

FIXED LENGTH INPUT: Enables the programmer to specify the allowable length of an input string. In addition it will allow a prompt string to be printed, print the number of underscores equal to the allowable length of the string, allow default strings to be specified, and finally, posesses some very nice editing features.

TIMED INPUT: Allows a specified amount of time to complete a response to an input command.

WINDOW: Quickly sets the text window boundaries in a single command. Great for screen management.

INVERTER: Changes the current screen window from normal to inverse text, or vice versa. Handy for highlighting text.

BORDER: Instantly create borders in the current text window with a single statement. Adds class to any program, and is easy to use.